Département TV numérique et Interactivité

MEDIAHIGHWAY
CONCEPTION DU SYSTEME MIDDLEWARE

DTI/MH+/serv/type/980000/1.0/JFB

05/06/98

version 1.0

HISTORIQUE

Date	Version	Auteur	Observations
05/09/97	0.0	JF Bertrand	Création du document.
04/06/98	1.0	JF Bertrand	Refonte du document en fonction des nouvelles spécifications du gestionnaire de téléchargement et du gestionnaire de FLASH.

SOMMAIRE

1. Gestionnaire de fichiers

1.1 Documents de référence.

[1] RFC1951	"DEFLATE Compressed Data Format Specification version 1.3"
[2]	MediaHighway+ : Gestionnaire de fichiers en FLASH.
[3]	MediaHighway+ : Gestionnaire de téléchargement.

1.2 Terminologie

Un volume représente l'organisation de l'ensemble des fichiers disponibles sur un support donné. Le support peut être local (mémoire RAM, FLASH ou ROM) ou externe (volume monté par l'intermédiaire d'un périphérique).

Un volume est organisé sous la forme d'un arbre avec un directory racine et des sous-directories. Le nombre de niveaux d'arborescence n'est pas limité. Chaque directory représente un nœud de l'arbre.

1.3 Le système de fichiers

1.3.1 Les volumes

Bien que le terminal ne possède pas de disque dur, un système de fichiers local est implémenté. Les fichiers sont stockés dans les différentes mémoires du terminal. Les fichiers sont organisés en volumes. Il existe un volume dans chaque support mémoire : RAM, FLASH et ROM. L'organisation des données est spécifique du support du volume.

Si le terminal ne comporte pas de ROM mais seulement de la mémoire FLASH, 2 volumes y sont stockés : le volume ROM est non modifiable tandis que le volume FLASH est modifiable.

Le système de fichiers peut également comporter un volume particulier appelé TRANSIENT, dont les données sont en RAM. Ce volume est utilisé pour implémenter la fonctionnalité "persistent storage" de la norme MHEG5. Une zone mémoire de taille fixe est réservée pour stocker les données des fichiers du volume. Cette zone est gérée comme une FIFO : Les fichiers les plus anciens sont supprimés au fur et à mesure pour faire place aux données des nouveaux fichiers stockés.

Un volume monté représente les fichiers disponibles sur un média de téléchargement. Ces fichiers peuvent être téléchargés dans le terminal. Ils peuvent éventuellement être transférés dans le volume RAM ou FLASH.

Au démarrage du terminal, le système de fichiers est dans l'état suivant :
- Le volume ROM est immédiatement disponible. Il décrit tous les fichiers résidents.
- La description de l'arborescence du volume FLASH est créée en RAM en fonction du contenu de la FLASH.
- Le volume RAM est vide.
- Le volume TRANSIENT est vide.
- Il n'existe aucun volume monté.

1.3.2 Les fichiers et les directories

Chaque fichier ou directory est décrit par les informations suivantes :

- Son nom.
- Son propriétaire. Il s'agit d'un nombre sur 2 octets représentant le code de l'opérateur (OPI) qui a généré le fichier.
- Ses droits d'accès en lecture et en écriture (3 fois 2 bits) pour :
  - Le propriétaire.
  - Les opérateurs appartenant au même groupe que le propriétaire.
  - Les opérateurs n'appartenant pas au même groupe.
- Des attributs :
  - Type de l'entité : fichier ou directory.
  - Mode de compression utilisé. Pour l'instant, les 2 options possibles sont : pas compressé ou compressé en utilisant le format "deflate".
  - Flag indiquant, dans le cas de données téléchargées, si les données associées à l'entité sont accessibles localement.
- Une référence vers les données. S'il s'agit d'un fichier, cette référence peut prendre diverses formes suivant que les données sont présentes dans le terminal ou non et suivant le support mémoire dans lequel elles se trouvent.

Tout accès à un fichier ou à un directory soit pour le lire, soit pour le modifier ou le supprimer est conditionné par les droits d'accès associés à l'entité. L'action de suppression est conditionnée par les droits d'accès en écriture.

Chaque fichier est référencé par son nom et par son chemin d'accès. Celui-ci est constitué de la concaténation des noms des différents directories constituant les nœuds de l'arborescence à parcourir pour trouver le fichier. Les noms sont séparés par le caractère "/".

Certains fichiers peuvent exister dans plusieurs volumes. On considère alors que la version du fichier située dans le volume le plus "volatile" masque la version du fichier contenue dans le volume le moins "volatile". Ainsi, un fichier en RAM masque le fichier du même nom situé dans le même directory en FLASH ou en ROM. De même, un fichier en FLASH masque le fichier du même nom situé dans le même directory en ROM. Un fichier peut également exister dans plusieurs volumes montés. On considère que les volumes montés sont plus volatiles que les volumes locaux. Le volume le plus volatile est le dernier volume monté.

Un fichier peut être compressé. L'algorithme de compression utilisé est l'algorithme "deflate" décrit dans le document [1]. Cette compression est indépendante du contenu sémantique du fichier.

1.3.3 Services proposés par le gestionnaire de fichier

- Ouverture d'un fichier en mode "read".
- Ouverture d'un fichier en mode "write", "append" ou "update" (uniquement sur les volumes RAM et FLASH).
- Lecture de données dans un fichier.
- Ecriture de données dans un fichier (uniquement sur les volumes RAM et FLASH).
- Fermeture d'un fichier.
- Création d'un directory.
- Copie de fichier ou de directory.
- Déplacement/renommage d'un fichier ou d'un directory.
- Suppression d'un fichier ou d'un directory.
- Fourniture d'informations sur un fichier.
- Montage/démontage d'un volume.
- Téléchargement de fichiers et de directories.

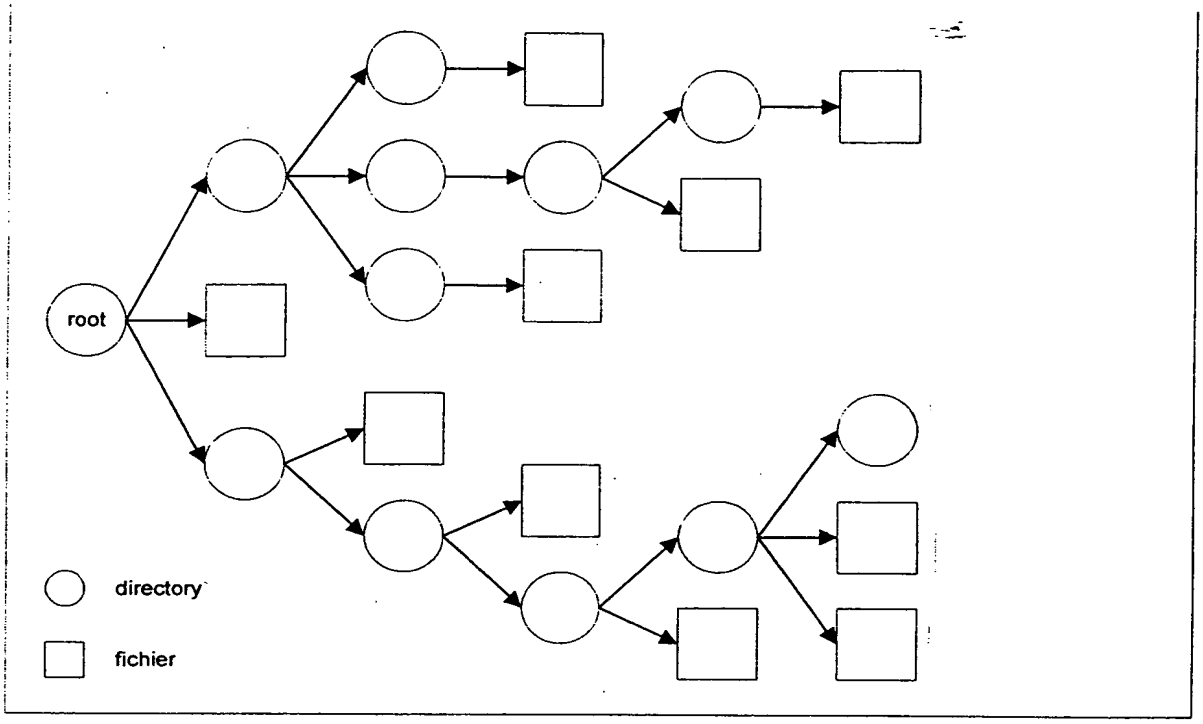

Figure 1 : Description de l'arborescence d'un volume.

1.3.4 Structures de description des fichiers et directories

La description de l'arborescence des directories et des fichiers est identique pour tous les volumes.
Les descripteurs des directories et des fichiers sont stockées :

- En ROM pour le volume ROM
- Dans le pool statique de la mémoire pour les autres volumes

Chaque volume contient une entité racine de type directory. Toutes ces entités sont elles-même contenues dans un directory racine général.

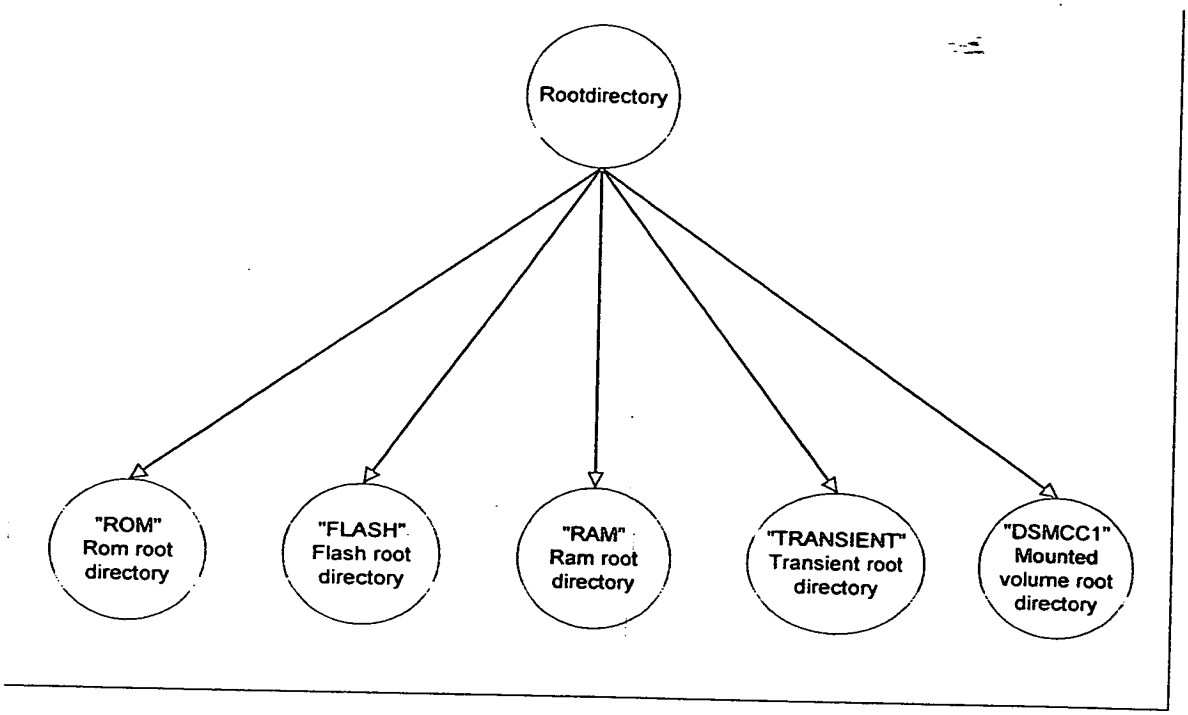

Figure 2 : Racine de l'arborescence.

1.3.4.1 Descripteur d'une entité de type directory

Champ	Type	Description
FsDirEntry {		
Parent	FsDirEntry *	Adresse du descripteur du directory auquel appartient l'entité
NameAddress	Char *	Adresse de la zone mémoire contenant le nom de l'entité.
Owner	Unsigned short	Numéro identifiant le propriétaire de l'entité.
Group	Unsigned char	Numéro identifiant le groupe auquel appartient le fichier.
AccessMode	Unsigned char	Droits d'accès associés.
Attributes	Unsigned char	Type et attributs de l'entité.
EntryCount	Unsigned short	Nombre d'entités contenues dans le directory.
EntryCapacity	Unsigned short	Nombre d'entités maximum que peut contenir la liste d'adresses des entités.
EntryList	FsEntry * [ ]	Liste de pointeurs sur les descripteurs des entités contenues dans le directory. Les pointeurs sont classés suivant l'ordre alphabétique des entités pointées.
blockCount	Unsigned short	Nombre de descripteurs des blocs de données des fichiers du directory.
blockCapacity	Unsigned short	Nombre maximum d'éléments que peut contenir la liste des descripteurs de blocs de données.
blockList	FileBlock[ ]	Liste des descripteurs de blocs de données des fichiers du directory.
}		

1.3.4.2 Descripteur d'une entité de type fichier

Champ	Type	Description
FsFileEntry {		
Parent	FsDirEntry *	Adresse du descripteur du directory auquel appartient l'entité
NameAddress	Char *	Adresse de la zone mémoire contenant le nom de l'entité.
Owner	Unsigned short	Numéro identifiant le propriétaire de l'entité.
Group	Unsigned char	Numéro identifiant le groupe auquel appartient le fichier.
AccessMode	Unsigned char	Droits d'accès associés.
Attributes	Unsigned char	Type et attributs de l'entité.
blockIndex	Unsigned short	Indice du descripteur du premier bloc de données du fichier dans la liste des descripteurs de blocs associée au directory auquel appartient le fichier.
blockCount	Unsigned short	Nombre de descripteurs des blocs de données du fichier.
}		

1.3.4.3 Descripteur d'un bloc de données d'un fichier

Champ	Type	Description
FileBlock {		
DataAddress	Unsigned Char *	Adresse de la zone mémoire contenant les données du bloc.
Size	Unsigned int	Taille du bloc de données en octets.
}		

1.4 Le volume ROM

Le volume ROM est constitué de tableaux et structures statiques. Les fichiers et les directories qu'il contient sont accessibles uniquement en lecture.

1.5 Le volume FLASH

La gestion des données dans le volume FLASH est décrite en détail dans le document [2].

1.5.1 Initialisation du volume FLASH

A l'initialisation du terminal, le gestionnaire de fichiers appelle le gestionnaire de FLASH pour lui demander de construire la description de son arborescence. Le gestionnaire de FLASH parcourt toutes les données stockées en FLASH et crée un descripteur pour chaque fichier et chaque directory.

1.5.2 Description en RAM de l'arborescence du volume FLASH

La description en mémoire contient des pointeurs directs sur les noms et sur les données des entités situées en FLASH. Par ailleurs, le gestionnaire de FLASH mémorise pour chaque entité stokée en FLASH l'adresse de son descripteur.

Chaque opération d'écriture en FLASH peut entraîner le déplacement d'autres données existantes (recopie d'une page dans une autre). Le gestionnaire de FLASH devra alors mettre à jour tous les pointeurs contenus dans les descripteurs des entités déplacées.

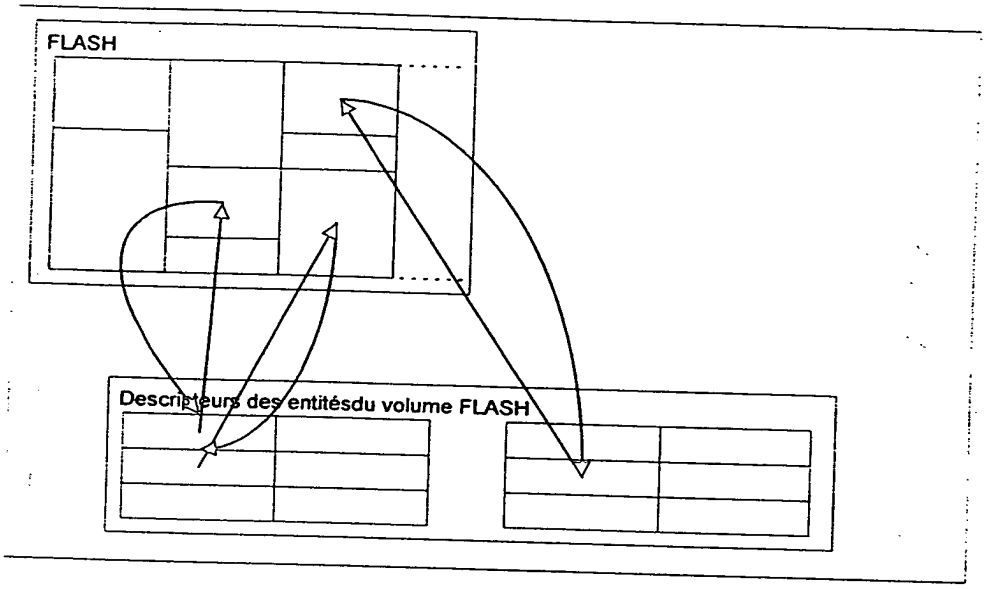

Figure 3 : Références croisées entre les données en FLASH et les descripteurs de ces données.

1.5.3 Modification du volume FLASH

Dans tous les cas, la modification du volume FLASH comporte plusieurs phases dont au moins une opération d'écriture asynchrone. Il est absolument impossible d'effectuer deux ou plusieurs modifications sur le volume FLASH en parallèle. Pendant une opération d'écriture en FLASH, la mémoire FLASH n'est pas accessible en lecture. L'accès à la FLASH est protégé par un moniteur.

1.6 Le volume RAM

Le volume RAM peut être utilisé pour stocker des données temporaires. Son contenu est perdu lorsqu'on éteint le décodeur. Des entités peuvent y être stockées par l'intermédiaires des opérations suivantes :

- Création de directory.
- Ouverture de fichier en écriture.
- Copie de fichiers ou d'arborescence à partir d'un autre volume.

1.7 Le volume TRANSIENT

Ce volume est un volume volatile qui est géré comme une FIFO. Le volume contient un seul niveau d'arborescence, c'est à dire qu'il contient uniquement un directory racine ("/TRANSIENT"). Ce directory contient uniquement des fichiers.

Une zone mémoire de taille fixe est réservée aux données des fichiers. La taille de cette zone est définie à l'initialisation du volume (fonction MhwFsInitPersistentStorage). Cette zone mémoire contient uniquement les données des fichiers. Les fichiers sont classés du plus ancien au plus récent. Lorsque la zone mémoire est pleine, les fichiers les plus anciens sont automatiquement supprimés (sans que l'application soit prévenue) pour faire de la place pour les nouveaux fichiers.

Les fichiers sont écrits en une seule opération grâce à une fonction d'interface particulière (MhwFsStorePersistent). Cette même fonction peut également être utilisée pour écraser le contenu d'un fichier existant. Un fichier dont le contenu est écrasé est reclassé comme étant le fichier le plus récent. Les fichiers peuvent être lus en utilisant la fonction MhwFsReadPersistent.

1.8 Volumes montés

Un volume monté représente des données téléchargées par l'intermédiaire d'un périphérique. Une fois téléchargées, les données sont décrites par une arborescence comme pour les volumes locaux. Les données des volumes montés sont accessibles en lecture uniquement.

Il existe 5 périphériques de téléchargement dans le terminal :
- démultiplexeur MPEG.
- port RS232.
- port centronics.
- lecteur de carte ISO 7816.
- modem

Pour accéder à un volume distant, une application doit demander le téléchargement des données du volume dans le terminal. Ce téléchargement s'effectue par étapes et dans un ordre qui n'est pas totalement quelconque. La première opération consiste à télécharger le directory racine du volume.

La gestion du téléchargement est décrite en détail dans le document [3].

1.8.1 Organisation d'un volume téléchargeable

Sur un média de téléchargement, les données d'un même volumes sont réparties en plusieurs blocs appelés "modules". Chaque module contient la description d'une arborescence. Le premier module du volume contient la description de la racine de l'arborescence du volume.

La figure suivante illustre l'organisation générale des données : Chaque module contient un descripteur pour chaque entité fichier ou directory. Les données associées à chaque entité peuvent être contenues soit dans le même module, soit dans un autre module.

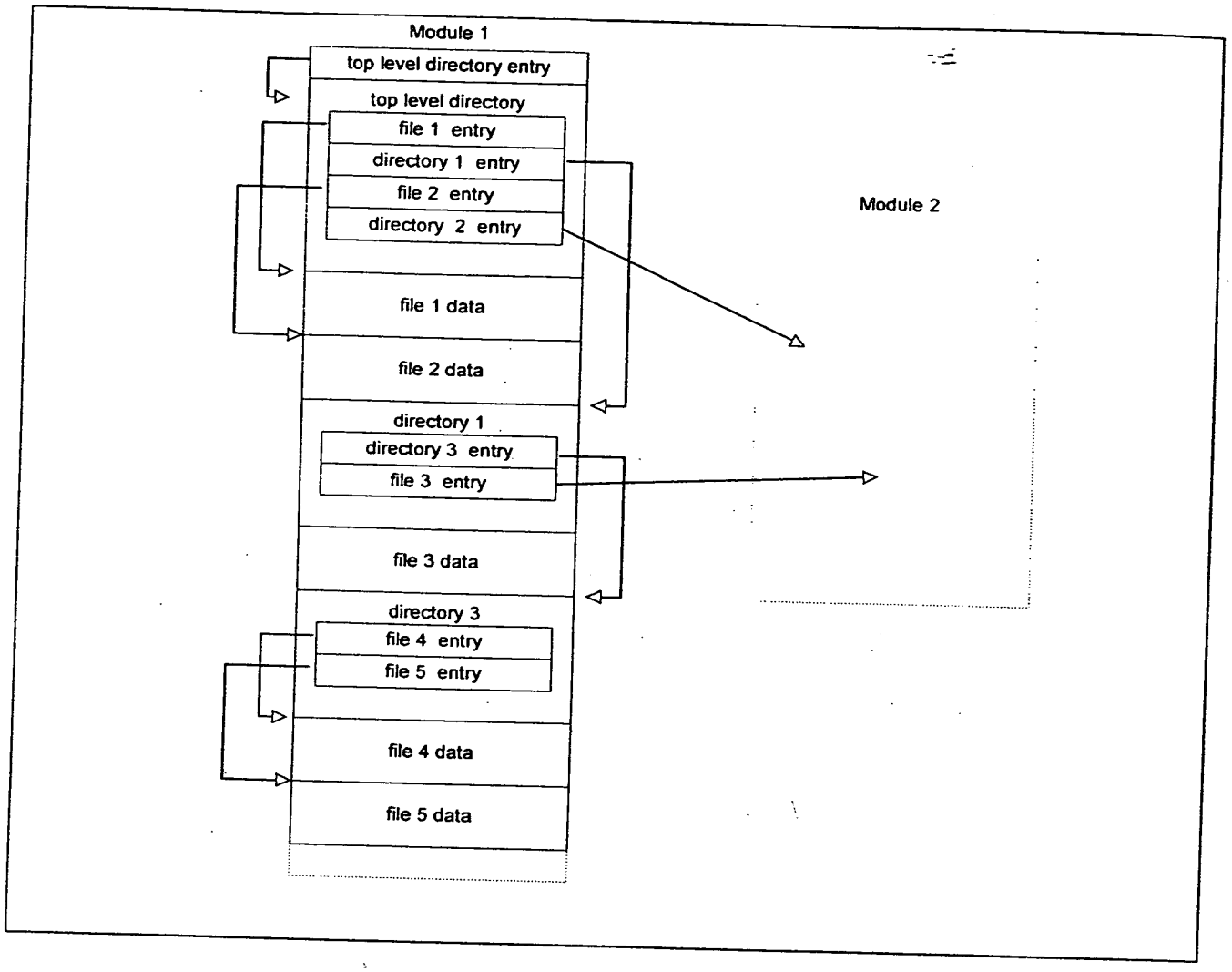

Figure 4 : Organisation des données sur le média de téléchargement.

1.8.2 Téléchargement de données

Le téléchargement est une opération asynchrone.

La première opération est l'opération de montage ou d'attachement. Elle permet de télécharger le premier module du volume. Ce module contient les directories et les fichiers constituant la racine du volume. Lorsque l'opération de téléchargement est terminée, ces entités sont contenues dans le directory racine du volume. Ce directory racine est créé par le gestionnaire

1.8.2.1 Montage d'un volume

Le montage d'un volume consiste à demander le téléchargement du premier module d'un volume sur un périphérique donné.

Sur le flot MPEG, il est possible de monter plusieurs volumes en parallèle. Sur les autres périphériques, on ne peut monter qu'un seul volume à la fois.

Dès la réception des données du module, les données sont analysées. On vérifie leur cohérence et éventuellement leur authenticité. On crée les descripteurs de toutes les entrées contenues dans le module pour constituer une arborescence.

1.8.2.2 Téléchargement d'un fichier ou d'un directory

Lorsqu'un volume a été monté, les entités du module reçu peuvent contenir des références vers d'autres modules.
Dès la réception des données du module, les données sont analysées. On vérifie leur cohérence et éventuellement leur authenticité. On crée les descripteurs de toutes les entrées contenues dans le module pour constituer une arborescence.

1.8.2.3 Démontage d'un volume

Le démontage d'un volume entraîne les actions suivantes :
- Suppression de la description de l'arborescence du volume.
- Suppression de toutes les données associées aux fichiers.
- Suppression du contexte de téléchargement.

1.9 Accès aux fichiers

Tous les fichiers situés dans un volume et dont les données sont présentes dans le terminal doivent être accessibles par l'application par l'intermédiaire d'une interface généralisée.

L'interface proposée est classique et contient les fonctions suivantes :
- Ouverture du fichier dans l'un des modes suivants :
  - Read
  - Write
  - Append
  - Update
- Lecture de données.
- Ecriture de données si le fichier est situé dans le volume RAM ou FLASH.
- Positionnement du pointeur de lecture/écriture.
- Fermeture du fichier.

Plusieurs versions du même fichier (même nom et même chemin d'accès) peuvent se trouver dans différents volumes à un instant donné. En général, l'application ne sait pas dans quel volume se trouve le fichier à charger. Le nom du fichier à rechercher est alors donné sans indication du volume (exemple : "stb/device/Mload.class"). Dans ce cas, le fichier doit être cherché dans les différents volumes accessibles en commençant par le dernier volume qui a été monté (chronologiquement), en continuant par les autres volumes montés puis le volume RAM, le volume FLASH et enfin le volume ROM. Dès que le fichier a été trouvé, la recherche s'arrête.

Il est toutefois possible d'indiquer précisément le volume dans lequel se trouve le fichier en ajoutant le nom du volume en tête du chemin. Les volumes locaux sont nommés "ROM", "FLASH" et "RAM". Chaque volume monté possède un nom qui pourra être donné par l'application ou attribué automatiquement (exemple "DSMCC1" ...). Pour indiquer que le chemin est donné en absolu, on le fera débuter par un caractère "/" (exemple : "/ROM/stb/device/Mload.class").

Lorsque le fichier est ouvert en écriture, il est bien sûr obligatoire d'indiquer le chemin absolu du directory dans lequel il doit être écrit (bien que seule soit autorisée l'écriture dans le volume RAM ou FLASH).

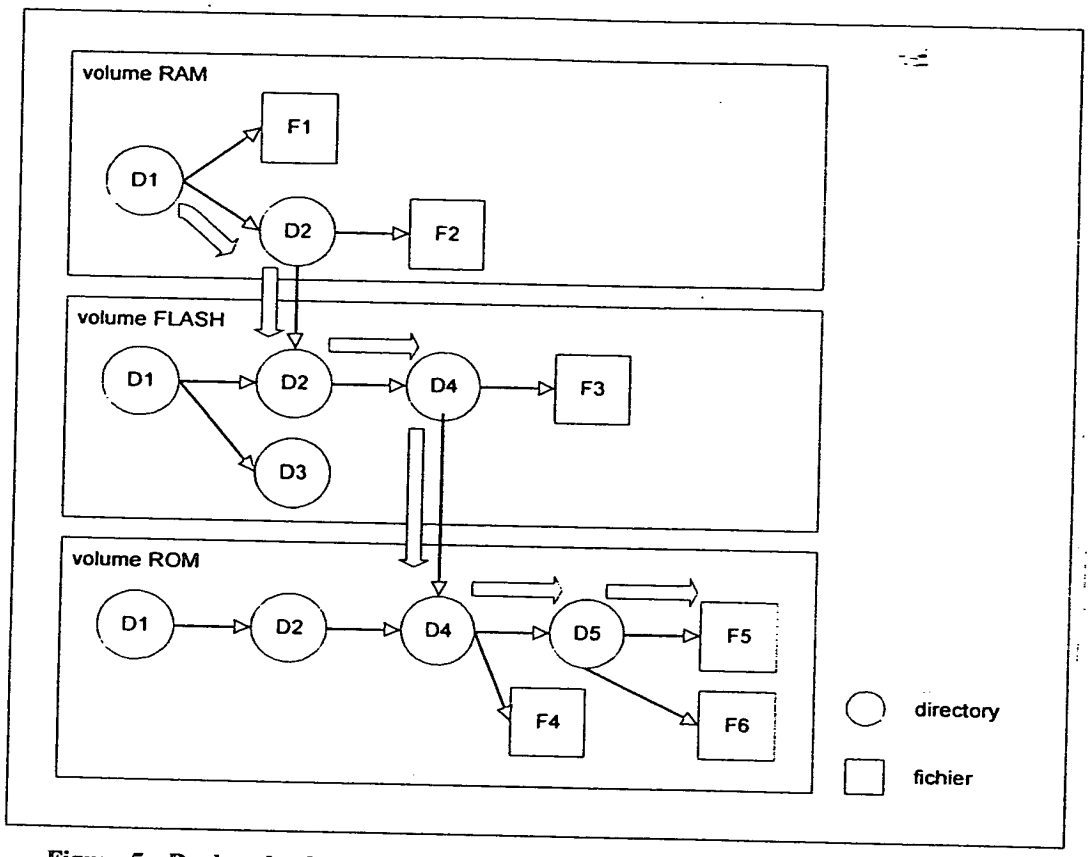

Figure 5 : Recherche du fichier "D1/D2/D4/D5/F5" au travers de différents volumes

1.9.1 Descripteur d'un flux de données

Champ	Type	Description
FileStream {		
Entry	FsEntry *	Adresse du descripteur du fichier.
OpenMode	Unsigned char	Mode d'ouverture du fichier.
Flags	Unsigned char	Flags internes.
StreamPointer	Unsigned int	Pointeur de lecture/écriture sur le flux externe.
CurrentBuffer	Unsigned char*	Adresse du bloc de données courant (flux interne).
currentBlockPointer	Unsigned int	Pointeur de lecture/écriture dans le bloc courant (flux interne).
currentBlockLength	Unsigned int	Taille en octets du bloc de données courant (flux interne).
currentBlockIndex	Unsigned short	Indice du bloc de données courant.
deviceContext	Void *	Adresse d'un contexte spécifique (utilisé par le gestionnaire de FLASH).
inflateContext	InflateCtx *	Adresse du contexte de décompression (flux en lecture uniquement).
}		

1.9.2 Ouverture d'un fichier en lecture

1.9.2.1 Flux de données en lecture

Une fois ouvert, un fichier est vu comme un flux de données. L'accès est en général séquentiel mais il est possible d'accéder directement à une position précise dans le flux de données.

Un contexte est créé pour contrôler le flux de données. Dans le cas où le fichier n'est pas compressé, il y a un seul flux de données (flux interne = flux externe). Dans le cas où les données sont compressés, il y a 2 flux de données : Le flux de données brut (flux interne) et le flux de données décompressées (flux externe). Dans ce dernier cas, un contexte supplémentaire est nécessaire pour gérer la décompression.

Dans un système de fichiers classique sur disque, les données sont en général lues bloc par bloc et stockées dans un buffer d'anticipation. Dans notre système de fichiers, dans le flux de données d'un fichier ouvert, à tout moment, un certain nombre d'octets sont disponibles directement et séquentiellement. Lorsque tous ces octets ont été lus par l'application, les données suivantes sont préparées.

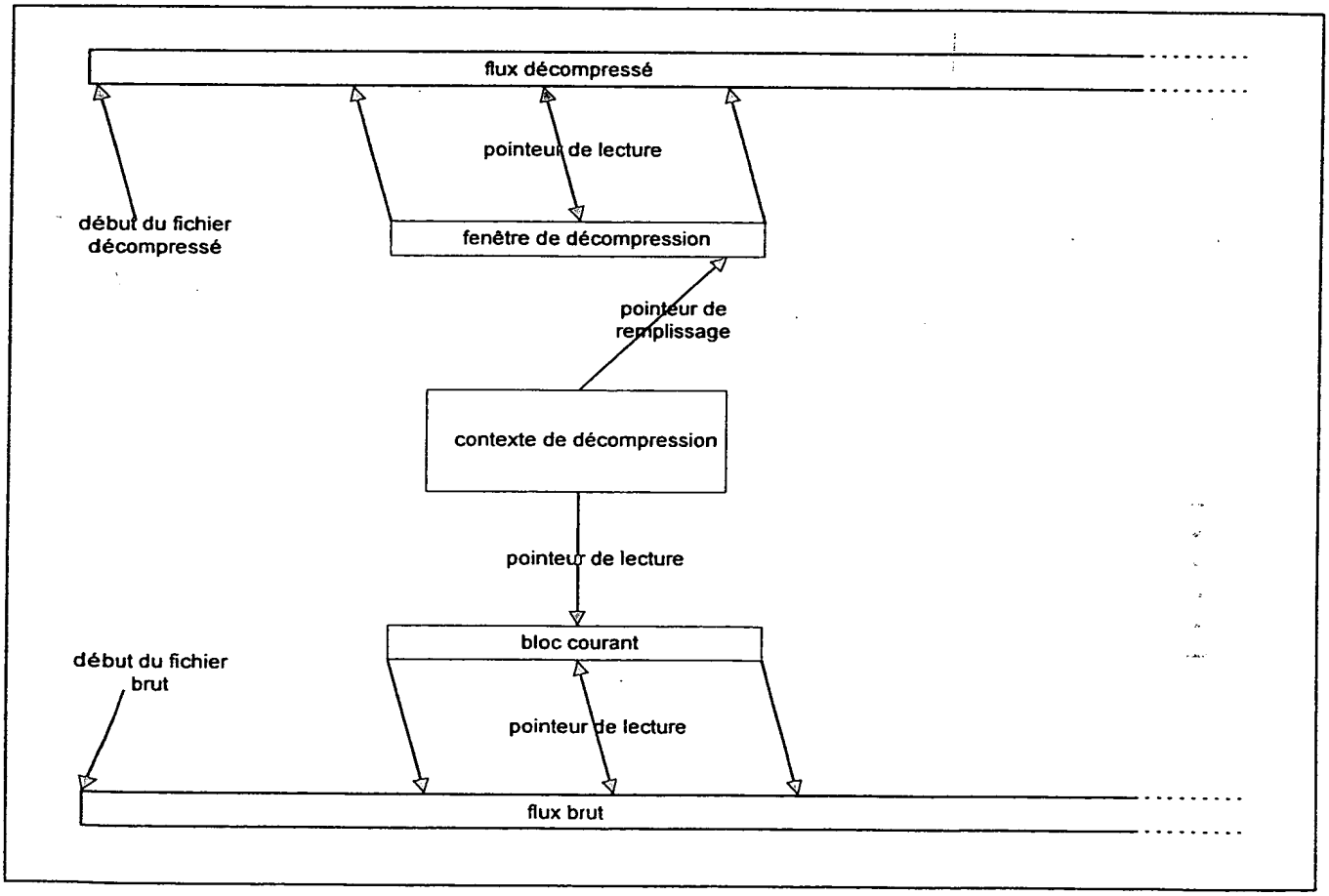

Figure 6 : Flux de données en lecture sur un fichier compressé.

1.9.3 Ouverture d'un fichier en écriture

Cette opération est possible uniquement sur le volume RAM ou sur le volume FLASH. Il est possible d'ouvrir un nouveau fichier ou un fichier existant. Dans les 2 cas, le chemin indiqué doit exister dans le volume FLASH. Le nom du chemin doit débuter par la chaîne "/RAM/" ou "/FLASH/".

Trois modes d'ouverture sont proposés :
- "write" : ouverture d'un nouveau fichier ou écrasement d'un fichier existant.
- "append" : écriture de données à la suite des données d'un fichier existant.
- "update" : remplacement de données ou ajout de nouvelles données dans un fichier existant.

Les modes "append" et "update" sont autorisés uniquement si le fichier existe déjà.

L'écriture avec compression n'est pas supportée.

1.9.3.1 Flux de données en écriture

L'écriture est bufferisée. La taille de chaque buffer est définie par une constante lorsque les données à écrire ne remplcent pas des données existantes. Sinon, la taille des blocs de données est celle des blocs existants.

Si le fichier est en FLASH, lorsque le buffer courant est plein, on alloue simplement un nouveau buffer.

Si le fichier est écrit dans le volume FLASH, on utilise un seul buffer en RAM. Lorsque le buffer est plein, son écriture en FLASH est déclenchée automatiquement. Le buffer se trouve ainsi vidé et prêt à recevoir de nouvelles données. L'écriture du buffer doit également être déclenchée par la fermeture du fichier s'il contient au moins un octet. Le fichier écrit en FLASH pourra ainsi être constitué de plusieurs blocs. Les références des blocs seront mémorisées dans le descripteur du directory destinataire au fur et à mesure de leur écriture.

1.9.3.2 Positionnement du pointeur d'écriture

Dans certains cas, l'application peut avoir besoin de positionner le pointeur d'écriture de façon non séquentielle. La nouvelle position peut être donnée en absolu par rapport au début ou à la fin du fichier ou relativement à la position courante du pointeur.

Si la nouvelle position indiquée correspond à un octet qui n'a pas encore été écrit (il se trouve par conséquent dans le buffer d'anticipation), l'opération consiste simplement à mettre à jour les pointeurs d'écriture relatif et absolu du flux.

Si la nouvelle position correspond à un octet qui a déjà été écrit, il faut dans l'ordre :
- Vider le contenu du buffer d'anticipation en écrivant en FLASH les octets qu'il contient.
- Remplir le buffer d'anticipation avec les octets du bloc correspondant si la position choisie correspond à des données existantes.
- Mettre à jour les pointeurs d'écriture relatif et absolu du flux.

1.10 Fonctions d'interface

1.10.1 MhwFsInit

SYNTAXE D'APPEL

int MhwFsInit (int aDebugLevel);

ARGUMENTS D'ENTREE

aDebugLevel

Niveau désiré de traces de mise au point (0 = sans traces).

DESCRIPTION

Cette fonction initialise le gestionnaire de fichiers.

1.10.2 MhwFsAttach

SYNTAXE D'APPEL

int MhwFsAttach (char * aVolume , Int32 aDevice, MhwDwnSGAddress *anSGAdress, MhwDwnModAddress *anSGModuleAddress, Int32 Authentication, MhwDwnVerParam *aMonitoringDescriptor);

ARGUMENTS D'ENTREE

aVolume

nom du volume à créer.

aDevice

Ce champ contient le port de communication physique où se trouve le Service Gateway à télécharger.
Les valeurs possibles pour ce champ sont:

DEV_MPEG
DEV_SERIE
DEV_PARALLELE
DEV_ISO7816
DEV_MODEM

anSGAddress

Ce champ contient l'adresse du Service Gateway sur lequel on va se connecter. La description complète est donnée dans la documentation de la fonction MhwDwnGCAttachGateway (Gestionnaire de téléchargement).

anSGModuleAddress

Ce champ contient les données nécessaires pour le chargement anticipé du module contenant le ServiceGateway. Si ce champ vaut NULL, le mode anticipé n'est pas utilisé. La description complète est donnée dans la documentation de la fonction MhwDwnGCAttachGateway (Gestionnaire de téléchargement).

Authentification

Ce champ contient un flag indiquant si l'on désire vérifier la signature des modules téléchargés.

AUTH_ON

La vérification de signature est activée. Tous modules sans signature ou avec une signature incorrecte seront rejetés.

AUTH_OFF

Les signatures ne sont pas vérifiées.

AMonitoringDescriptor

Ce champ décrit le mode de surveillance des versions des modules. La description complète est donnée dans la documentation de la fonction MhwDwnGCAttachGateway (Gestionnaire de téléchargement).

VALEUR DE RETOUR

Si l'opération est acceptée, la fonction rend un identifiant de chargement positif ou nul.
Si l'opération est refusée, la fonction rend l'un des codes d'erreur suivants :
ERR_FS_MEMORY : Echec d'allocation mémoire.
ERR_FS_ALREADY_ATTACHED : Il existe déjà un volume portant ce nom.

Autres codes d'erreur rendus par la fonction MhwDwnGCAttachGateway.

DESCRIPTION

Cette fonction télécharge la première unité de téléchargement d'un volume. Cette unité contient le directory général du volume. Elle peut également contenir des fichiers et des sous-directories. La fonction crée en mémoire l'arborescence correspondante après avoir vérifié la cohérence et éventuellement authentifié les données. Un nom doit être donné au volume par l'appelant. Les noms "ROM", "RAM", "TRANSIENT" et "FLASH" sont interdits. Le nom ne doit contenir aucun caractère '/'.
L'opération est asynchrone. Elle engendre le chargement du module DSMCC contenant le service gateway. Un événement de type MHW_EVT_FSATTACH_TYPE est généré pour indiquer la fin de l'opération demandée.

Le code de l'événement est l'identifiant de chargement rendu par la fonction.

L'événement contient un code d'erreur qui vaut :
ERR_NONE : L'opération s'est déroulée correctement.
ERR_FS_LOAD_FAILED : Le chargement n'a pu être effectué (timeout ou manque de ressource physique).

1.10.3 MhwFsDetach

SYNTAXE D'APPEL

int MhwFsDetach(char * aVolume);

ARGUMENTS D'ENTREE

aVolume

nom du volume à démonter.

DESCRIPTION

Cette fonction démonte un volume en supprimant le directory racine du volume, tous ses sous-directories et les fichiers qu'ils contiennent. Toutes les données référencées sont également supprimées de la mémoire. L'opération est synchrone. Si le volume indiqué est en cours de montage (opération MhwFsAttach en cours), la demande d'attachement est abortée.

VALEUR DE RETOUR

ERR_NONE : L'opération est acceptée.
ERR_FS_NOT_ATTACHED : Le volume indiqué n'existe pas.
Autres codes d'erreurs rendus par la fonction MhwDwnGCDetachGateway.

1.10.4 MhwFsDownload

SYNTAXE D'APPEL

int MhwFsDownload (char * aPath);

ARGUMENTS D'ENTREE

aPath

chemin d'accès du fichier ou du directory. Le chemin d'accès peut être donné en relatif ou en absolu.

VALEUR DE RETOUR

Si l'opération est acceptée, la fonction rend un identifiant de téléchargement positif ou nul.
Si l'opération est refusée, la fonction rend l'un des codes d'erreur suivants :
ERR_FS_MEMORY : Echec d'allocation mémoire.
ERR_FS_PATH_NOT_FOUND : Le chemin indiqué ne correspond pas à un fichier existant.
ERR_FS_ALREADY_LOADED : L'entité a déjà été téléchargée.
Autres codes d'erreur rendus par la fonction MhwDwnGCGetObject.

DESCRIPTION

Cette fonction permet de demander le téléchargement d'un fichier ou d'un directory.

L'opération est asynchrone. Elle peut engendrer un ou plusieurs chargements de modules pour accéder aux données. Un événement de type MHW_EVT_FSOPEN_TYPE est généré pour indiquer la fin de l'opération demandée. Le code de l'événement est l'identifiant rendu par la fonction.

L'événement contient un code d'erreur qui vaut :
ERR_NONE : si l'opération s'est déroulée correctement.
ERR_FS_LOAD_FAILED : Le chargement n'a pu être effectué (timeout ou manque de ressource physique).

1.10.5 MhwFsLoadAbort

SYNTAXE D'APPEL

int MhwFsLoadAbort (Int32 Id);

ARGUMENTS D'ENTREE

Id

Identifiant de chargement rendu par la fonction MhwFsDownload.

VALEUR DE RETOUR

En retour, la fonction rend :
ERR_NONE : L'opération est acceptée.
ERR_FS_NO_SUCH_LOADING : L'identifiant de chargement est inconnu.
Autres codes d'erreur rendus par la fonction MhwDwnGCAbort.

DESCRIPTION

Cette fonction permet d'aborter une demande de téléchargement initiée par l'appel de la fonction MhwFsDownload.

1.10.6 MhwFsRemove

SYNTAXE D'APPEL

int MhwFsRemove (char * aPath);

ARGUMENTS D'ENTREE

aPath

chemin du fichier ou directory à supprimer. Le chemin doit débuter par le caractère '/', suivi du nom du volume (exemple "/RAM/...."). Il est impossible de supprimer un fichier ou un directory dans le volume ROM.

VALEUR DE RETOUR

ERR_NONE : pas d'erreur
ERR_FS_PATH_NOT_FOUND : le chemin d'accès n'existe pas.
ERR_FS_ACCESS_DENIED : l'accès en écriture au directory parent ou à l'un des sous-directories est refusé.

DESCRIPTION

Cette fonction supprime le fichier ou le directory dont on donne le chemin d'accès. Elle supprime également tous les sous-directories si le chemin indiqué représente un directory. Le chemin doit être donné en absolu et débuter par le nom du volume contenant le fichier ou directory. Il peut s'agir du volume RAM, du volume FLASH ou d'un volume monté.

1.10.7 MhwFsCreateDirectory

SYNTAXE D'APPEL

int MhwFsCreateDirectory (char * aPath);

ARGUMENTS D'ENTREE

aPath

chemin du directory à créer. Le chemin doit débuter par le caractère '/', suivi du nom du volume (exemple "/RAM/...."). Il est impossible de créer un directory dans le volume ROM.

VALEUR DE RETOUR

ERR_NONE : pas d'erreur
ERR_FS_PATH_NOT_FOUND : le chemin d'accès du directory parent n'existe pas.
ERR_FS_ACCESS_DENIED : l'accès en écriture au directory parent est refusé.
ERR_FS_NOT_A_DIRECTORY : le chemin d'accès représentant le directory parent ne correspond pas à un directory mais à un fichier.

DESCRIPTION

Cette fonction crée le directory dont on donne le nom. Le chemin du directory doit être donné en absolu et débuter par le nom du volume contenant le directory. Il peut s'agir du volume RAM ou du volume FLASH. Tous les directories intermédiaires mentionnés dans le chemin d'accès doivent exister.

1.10.8 MhwFsCopy

SYNTAXE D'APPEL

int MhwFsCopy (char * aSourcePath , char * aDestinationPath);

ARGUMENTS D'ENTREE

aSourcePath

chemin actuel du fichier ou directory. Le chemin doit débuter par le caractère '/', suivi du nom du volume (exemple "/RAM/...."). Il peut représenter un fichier ou un directory situé dans n'importe quel volume.

aDestinationPath

chemin du fichier ou directory destination. Le chemin doit débuter par le caractère '/', suivi du nom du volume (exemple "/RAM/...."). Il est impossible de copier un fichier ou un directory dans le volume ROM. Le chemin doit représenter soit un directory existant, soit un fichier ou un directory non existant dans un directory existant.

VALEUR DE RETOUR

ERR_NONE : pas d'erreur
ERR_FS_PATH_NOT_FOUND : le chemin d'accès source et/ou destination n'existe pas.
ERR_FS_ACCESS_DENIED : accès en écriture refusé sur le directory destination ou accès en lecture refusé sur le directory source.
ERR_FS_MEMORY : échec d'allocation mémoire pour la copie.
ERR_FS_NOT_A_DIRECTORY : Le chemin destination représente un fichier existant.

DESCRIPTION

Cette fonction copie un fichier ou un directory et toute l'arborescence qui en dépend.

Le chemin destination doit représenter un directory existant ou une entrée non existante dans un directory existant.

Les chemins doivent être donnés en absolu. L'opération est permise si le chemin destination est situé dans le volume RAM ou dans le volume FLASH.

1.10.9 MhwFsMove

SYNTAXE D'APPEL

int MhwFsMove (char * aSourcePath , char * aDestinationPath);

ARGUMENTS D'ENTREE

aDirectory

chemin actuel du fichier ou directory. Le chemin doit débuter par le caractère '/', suivi du nom du volume (exemple "/RAM/....").

aDestinationPath

chemin du fichier ou directory destination. Le chemin doit débuter par le caractère '/', suivi du nom du volume (exemple "/RAM/....").

VALEUR DE RETOUR

ERR_NONE : pas d'erreur
ERR_FS_PATH_NOT_FOUND : le chemin d'accès source et/ou destination n'existe pas.
ERR_FS_ACCESS_DENIED : accès en écriture refusé sur le directory destination ou sur le directory source.
ERR_FS_MEMORY : échec d'allocation mémoire.
ERR_FS_DUPLICATE_NAME : Il existe déjà une entrée du même nom dans le directory destination.

DESCRIPTION

Cette fonction déplace un fichier ou un directory et toute l'arborescence qui en dépend.

Le chemin destination doit représenter un directory existant ou une entrée non existante dans un directory existant. S'il existe déja une entrée du même nom dans le directory destination, l'opération est refusée. Si les directories source et destination sont identiques, l'opération se résume à un changement de nom. Sinon, l'entrée représentée par le chemin source est retirée du directory qui la contient pour être ajoutée au directory destination.

Les chemins doivent être donnés en absolu. Les directories source et destination doivent être situés dans le même volume.L'opération est permise uniquement dans le volume RAM et dans le volume FLASH.

1.10.10 MhwFsGetFileList

SYNTAXE D'APPEL

int MhwFsGetFileList (char * aPath , char *aFilter , int aMode , char ** aList);

ARGUMENTS D'ENTREE

aPath

chemin du directory. Le chemin peut être donné en absolu ou en relatif.

aFilter

filtre de sélection des fichiers. Il s'agit d'une chaîne de caractères qui peut contenir le caractère '*' qui représente un nombre quelconque d'occurrences de n'importe quel caractère et le caractère '?' qui représente une occurrence de n'importe quel caractère. Si la valeur du paramètre est un pointeur nul ou une chaîne vide, aucun filtrage n'est effectué sur les noms.
Exemples : "*.class" , "*" , "*.*" , "ab?d.e?".

aMode

ce paramètre indique si les extensions doivent être conservées dans la liste des noms sélectionnés. Il peut prendre l'une des 2 valeurs suivantes :
MHW_FS_KEEP_EXTENSIONS : les extensions sont conservées.
MHW_FS_DONT_KEEP_EXTENSIONS : les extensions ne sont pas conservées.

ARGUMENTS DE SORTIE

aList

buffer alloué par la fonction contenant la liste des noms des fichiers ou sous-directories du directory séparés par des line-feed (code ascii 10). En cas d'erreur ou dans le cas où la liste est vide (c'est à dire si le directory ne contient aucun fichier dont le nom satisfait le critère spécifié par le filtre), la valeur du paramètre est NULL.

VALEUR DE RETOUR

Si pas d'erreur : Nombre de fichiers composant la liste.

ERR_FS_PATH_NOT_FOUND : le chemin d'accès n'existe pas ou ne représente pas un directory.
ERR_FS_ACCESS_DENIED : accès en lecture refusé sur le directory.
ERR_FS_MEMORY : échec d'allocation mémoire.

DESCRIPTION

Cette fonction rend une liste de noms de fichiers/directories sélectionnés dans un directory. La fonction rend l'adresse d'un buffer qu'elle a alloué. Le buffer est alloué uniquement si le nombre de fichiers trouvés est strictement positif. La libération de ce buffer est à la charge de l'appelant. Le buffer contient les noms des fichiers ou sous-directories en ascii. Chaque nom est terminé par le caractère "line-feed" à l'exception du dernier qui est terminé par un caractère nul. La fonction rend le nombre de fichiers trouvés.

1.10.11 MhwFsFileExist

SYNTAXE D'APPEL

boolean MhwFsFileExist (char * aPath);

ARGUMENTS D'ENTREE

aPath

chemin d'accès du fichier ou du directory. Le chemin peut être donné en relatif ou en absolu.

VALEUR DE RETOUR

TRUE si le fichier/directory existe, FALSE sinon.

DESCRIPTION

Cette fonction indique si un fichier ou un directory existe. Le chemin peut ou non contenir la référence du volume dans lequel on doit chercher le fichier.

1.10.12 MhwFsIsFile

SYNTAXE D'APPEL

boolean MhwFsIsFile (char * aPath);

ARGUMENTS D'ENTREE

aPath

chemin d'accès du fichier ou du directory. Le chemin peut être donné en relatif ou en absolu.

VALEUR DE RETOUR

TRUE si le chemin représente un fichier existant, FALSE sinon.

DESCRIPTION

Cette fonction indique si un chemin d'accès représente un fichier. Le chemin peut ou non contenir la référence du volume dans lequel on doit chercher le fichier.

1.10.13 MhwFsIsDirectory

SYNTAXE D'APPEL

boolean MhwFsIsDirectory (char * aPath);

ARGUMENTS D'ENTREE

aPath

chemin d'accès du fichier ou du directory. Le chemin peut être donné en relatif ou en absolu.

VALEUR DE RETOUR

TRUE si le chemin représente un directory existant, FALSE sinon.

DESCRIPTION

Cette fonction indique si un chemin d'accès représente un directory. Le chemin peut ou non contenir la référence du volume dans lequel on doit chercher le directory.

1.10.14 MhwFsIsAccessible

SYNTAXE D'APPEL

boolean MhwFsIsAccessible (char * aPath , Card8 anAccessMode);

ARGUMENTS D'ENTREE

aPath

chemin d'accès du fichier ou du directory. Le chemin peut être donné en relatif ou en absolu.

anAccessMode

Mode d'accès :
MHW_FS_READ : accès en lecture.
MHW_FS_WRITE : accès en écriture.

VALEUR DE RETOUR

TRUE si le thread courant possède les droits d'accès sur le fichier/directory, FALSE sinon.

DESCRIPTION

Cette fonction indique si le thread courant possède les droits d'accès en lecture ou en écriture sur un fichier ou un directory. Le chemin peut ou non contenir la référence du volume dans lequel on doit chercher le directory.

1.10.15 MhwFsGetFileLength

SYNTAXE D'APPEL

Int32 MhwFsGetFileLength (char * aPath);

ARGUMENTS D'ENTREE

aPath

chemin d'accès du fichier. Le chemin peut être donné en relatif ou en absolu.

VALEUR DE RETOUR

taille du fichier si celui-ci existe.
ERR_FS_PATH_NOT_FOUND : le chemin d'accès n'existe pas ou ne représente pas un fichier.
ERR_FS_ACCESS_DENIED : accès en lecture refusé sur le directory contenant le fichier.

DESCRIPTION

Cette fonction rend la taille en octets d'un fichier.

1.10.16 MhwFsFileStreamOpen

SYNTAXE D'APPEL

int MhwFsFileStreamOpen (char * aPath, int aMode);

ARGUMENTS D'ENTREE

aPath

chemin d'accès du fichier. Le chemin d'accès peut être donné en relatif s'il s'agit d'un fichier existant. Il doit être donné obligatoirement en absolu s'il s'agit d'un fichier à créer.

aMode

mode d'ouverture du fichier :
MHW_FS_READ : lecture de données.
MHW_FS_WRITE : écriture de données.

MHW_FS_APPEND : ajout de données.
MHW_FS_UPDATE : lecture, remplacement et ajout de données.

DESCRIPTION

Cette fonction permet d'ouvrir ou de créer un fichier. La création de fichier est autorisée uniquement en mode MHW_FS_WRITE. En cas de création de fichier, le chemin doit être donné en absolu, c'est à dire débuter par un caractère '/' suivi du nom du volume (exemple : "/ROM/java/lang/Thread.class"). Si le fichier est ouvert en lecture, le chemin peut être donné relativement au volume dans lequel il se trouve (exemple : "java/lang/Thread.class"). Dans ce cas, le fichier est recherché dans tous les volumes disponibles, du plus volatile (dernier volume monté) au moins volatile (volume ROM). En retour, la fonction rend l'identifiant du descripteur représentant le flux de données ou -1 en cas d'échec.

En cas d'échec de l'opération, l'erreur globale est positionnée à l'une des valeurs suivantes :
ERR_FS_PATH_NOT_FOUND : le chemin indiqué ne correspond pas à un fichier existant ou , dans le cas du mode d'ouverture MHW_FS_WRITE, le chemin ne correspond pas à une entrée inexistante dans un directory existant.
ERR_FS_ACCESS_DENIED : Le thread courant ne possède pas les droits en lecture ou en écriture sur le fichier, ou, dans le cas d'une création de fichier, le thread courant ne possède pas les droits d'écriture sur le directory dans lequel on demande à créer le fichier.

1.10.17 MhwFsFileStreamClose

SYNTAXE D'APPEL

int MhwFsFileStreamClose (int aFileDescriptor);

ARGUMENTS D'ENTREE

aFileDescriptor

descripteur du flux de données.

DESCRIPTION

Cette fonction permet de fermer un flux de données sur un fichier. Si le fichier a été ouvert en écriture, les données contenues dans le buffer de blocage sont flushées.

1.10.18 MhwFsFileStreamRead

SYNTAXE D'APPEL

Int32 MhwFsFileStreamRead (int aFileDescriptor , Card8 aBuffer[] , Int32 aSize);

ARGUMENTS D'ENTREE

aFileDescriptor

descripteur du flux de données.

aBuffer

adresse du buffer dans lequel doivent être stockées les données lues.

aSize

nombre d'octets à lire.

VALEUR DE RETOUR

La fonction rend le nombre d'octets effectivement lus. Ce nombre peut être inférieur au nombre d'octets demandés si la fin du fichier est atteinte. Ce nombre vaut –1 si le descripteur ne correspond pas à un fichier ouvert en lecture.

DESCRIPTION

Cette fonction permet de lire des données dans un fichier ouvert en lecture.

1.10.19 MhwFsFileStreamWrite

SYNTAXE D'APPEL

int MhwFsFileStreamWrite (int aFileDescriptor , Card8 aBuffer[] , Int32 aSize);

ARGUMENTS D'ENTREE

aFileDescriptor

descripteur du flux de données.

aBuffer

adresse du buffer contenant les données à écrire.

aSize

nombre d'octets à écrire.

VALEUR DE RETOUR

Le code retour de la fonction est nul si l'opération s'est déroulée correctement. Il vaut –1 si l'opération est refusée ou si elle a échoué.

DESCRIPTION

Cette fonction permet d'écrire des données dans un fichier ouvert en écriture.

1.10.20 MhwFsFileStreamSeek

SYNTAXE D'APPEL

Int32 MhwFsFileStreamSeek (int aFileDescriptor , Int32 anOffset , int where);

ARGUMENTS D'ENTREE

aFileDescriptor

descripteur du flux de données.

anOffset

offset auquel doit être positionné le pointeur courant du flux de données.

aMode

Mode indiquant comment doit être interprété l'offset :
MHW_FS_BEGINNING : l'offset est indiqué par rapport au début du fichier.
MHW_FS_CURRENT : l'offset est indiqué par rapport à la valeur actuelle du pointeur courant.
MHW_FS_END : l'offset est indiqué par rapport à la fin du fichier. Sa valeur doit être négative.

VALEUR DE RETOUR

La fonction rend la valeur du pointeur courant après l'opération ou –1 si l'un des paramètres d'entrée a une valeur erronée ou si on tente de positionner le pointeur courant à une valeur négative.

DESCRIPTION

Cette fonction permet de positionner le pointeur courant d'un flux de données. Elle est autorisée uniquement si le fichier a été ouvert en mode MHW_FS_READ ou en mode MHW_FS_UPDATE. S'il s'agit d'un flux en lecture et si la position demandée ne correspond pas à un octet contenu dans le buffer courant, l'opération peut entraîner la lecture et éventuellement la décompression de nouvelles données. S'il s'agit d'un flux en écriture et si la position demandée ne correspond pas à un octet compris dans le buffer de blocage, le contenu du buffer est écrit dans le fichier puis le buffer de blocage est rempli avec les données du bloc correspondant à la position demandée.

1.10.21 MhwFsFileStreamTell

SYNTAXE D'APPEL

Int32 MhwFsFileStreamTell (int aFileDescriptor);

ARGUMENTS D'ENTREE

aFileDescriptor

descripteur du flux de données.

VALEUR DE RETOUR

La fonction rend la valeur du pointeur courant du flux de données.

DESCRIPTION

Cette fonction rend la valeur du pointeur courant d'un flux de données, c'est à dire la position par rapport au début du fichier du prochain octet à lire ou à écrire.

1.10.22 MhwFsFileStreamGetLength

SYNTAXE D'APPEL

Int32 MhwFsFileStreamGetLength (int aFileDescriptor);

ARGUMENTS D'ENTREE

aFileDescriptor

descripteur du flux de données.

VALEUR DE RETOUR

La fonction rend la longueur totale du flux de données.

DESCRIPTION

Cette fonction rend la longueur totale en octets d'un flux de données.

1.10.23 MhwFsInitPersistentStorage

SYNTAXE D'APPEL

Int32 MhwFsInitPersistentStorage (Int32 aSize);

ARGUMENTS D'ENTREE

aSize

Taille maximale en octets réservée aux données des fichiers "persistent storage".

VALEUR DE RETOUR

ERR_NONE : l'opération est acceptée.
ERR_FS_MEMORY : Echec d'allocation mémoire.
ERR_FS_DUPLICATE_NAME : L'initialisation a déjà été effectuée.

DESCRIPTION

Cette fonction permet de limiter la taille mémoire occupée par les données du volume "persistent storage". Le gestionnaire de fichier gère alors cet espace en supprimant automatiquement les fichiers les plus anciennement accédés lorsque la limite est atteinte. Cette fonction doit être appelée une seule fois à l'initialisation du système.

1.10.24 MhwFsStorePersistent

SYNTAXE D'APPEL

Int32 MhwFsStorePersistent (char *aName, Card8 aBuffer[] , Int32 aSize);

ARGUMENTS D'ENTREE

aName

nom du fichier.

aBuffer

adresse du buffer contenant les données à écrire.

aSize

nombre d'octets à écrire.

VALEUR DE RETOUR

ERR_NONE : l'opération est acceptée.
ERR_FS_MEMORY : Echec d'allocation mémoire (pour le descripteur du fichier).

DESCRIPTION

Cette fonction permet d'écrire des données dans un fichier. Le fichier est créé dans le volume "persistent storage". Si le fichier n'existe pas, il est créé. Si le fichier existe, les données qu'il contient sont remplacées par les nouvelles données. La taille des nouvelles données peut être différente de celle des données existantes. Le nom du fichier ne doit pas contenir le caractère '/'.

1.10.25 MhwFsReadPersistent

SYNTAXE D'APPEL

Int32 MhwFsReadPersistent (char *aName, Card8 aBuffer[] , Int32 aSize);

ARGUMENTS D'ENTREE

aName
nom du fichier.
aBuffer
adresse du buffer pour stocker les données lues.
aSize
nombre d'octets à lire.

VALEUR DE RETOUR

ERR_NONE : l'opération est acceptée.
ERR_FS_PATH_NOT_FOUND : Le fichier spécifié n'existe pas.

DESCRIPTION

Cette fonction permet de lire des données dans un fichier du volume "persistent storage".

1.10.26 MhwFsGetFileEntry

SYNTAXE D'APPEL

FsEntry * MhwFsGetFileEntry (char *aPath);

ARGUMENTS D'ENTREE

aPath
chemin d'accès au fichier.

VALEUR DE RETOUR

La fonction rend l'adresse de la structure décrivant le fichier ou NULL si le chemin indiqué ne correspond pas à un fichier existant.

DESCRIPTION

Cette fonction rend l'adresse du descripteur d'un fichier.

1.10.27 MhwFsGetEntryBlockCount

SYNTAXE D'APPEL

Int32 MhwFsGetEntryBlockCount (FsEntry *anEntry);

ARGUMENTS D'ENTREE

anEntry
adresse du descripteur du fichier.

VALEUR DE RETOUR
La fonction rend le nombre de blocs de données constituant le fichier ou –1 si l'adresse passée en entrée est invalide.

DESCRIPTION
Cette fonction rend le nombre de blocs de données constituant un fichier.

1.10.28 MhwFsGetEntryBlockList

SYNTAXE D'APPEL
Int32 MhwFsGetEntryBlockList (FsEntry *entry, FileBlock blocks[], Int32 blockIndex, Int32 blockCount);

ARGUMENTS D'ENTREE
entry
adresse du descripteur du fichier.
Blocks
Tableau de structures FileBlock.
blockIndex
Indice du premier bloc.
blockCount
Nombre de blocs.

ARGUMENTS DE SORTIE
blocks
Le tableau est rempli avec la description des blocs de données du fichier d'indice `blockIndex` à `blockIndex+blockCount-1`.

VALEUR DE RETOUR
La fonction rend 0 si l'opération s'est déroulée correctement ou –1 si l'opération est refusée ou si elle a échoué.

DESCRIPTION
Cette fonction rend la description de blocs de données constituant un fichier. La structure `FileBlock` est constituée de la façon suivante :

```
typedef struct
{
     Card8 *ptr;
     Card32 length;
} FileBlock;
```